Machine intelligence

7th lecture

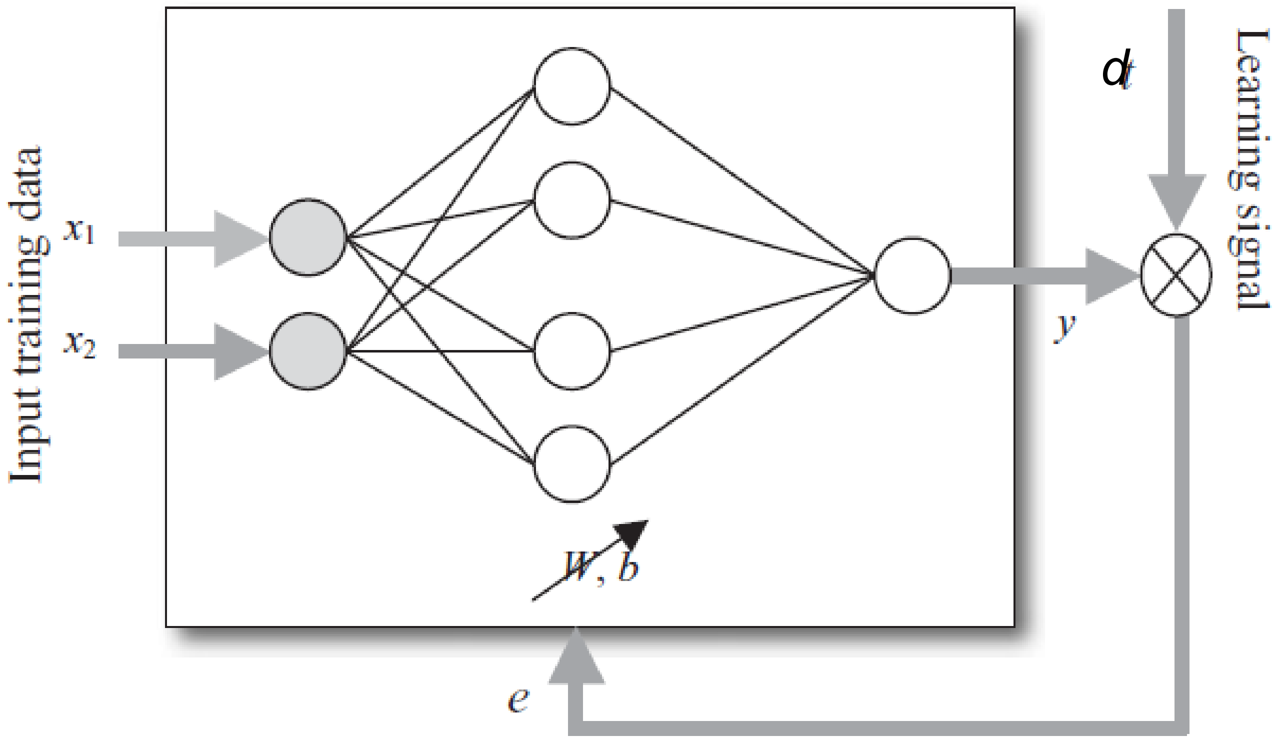
Widrow-Hoff learning and Delta Rule learning

Dr. Ahmad Al-Mahasneh

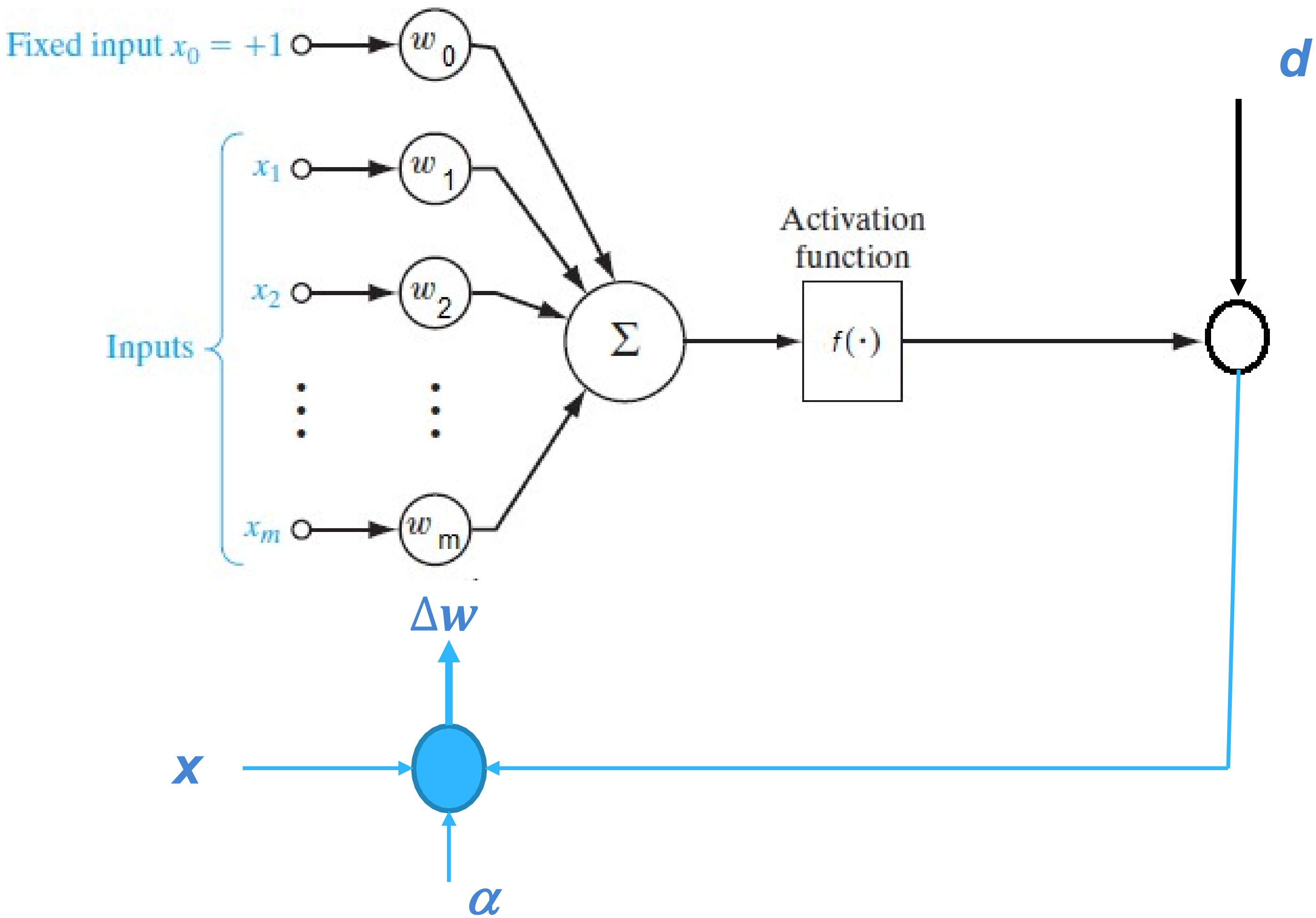
OUTLINE

- Widrow-Hoff Algorithm
- Perceptron example
- Delta Rule Learning (one neuron)
- Example
- MATLAB example
- Delta Rule Learning (multi-neurons)

Supervised Learning



WIDROW-HOFF LEARNING ALGORITHM



WIDROW-HOFF LEARNING ALGORITHM

1. $net = \sum_{i=1}^{m} w_i x_i$

2. Using a linear activation function $y = f(net) = net$

3. The error between the network output and the desired output is:

$$e = \frac{1}{2}(d - y)^2$$

4. Using the chain rule, the derivative w.r.t. weights is:

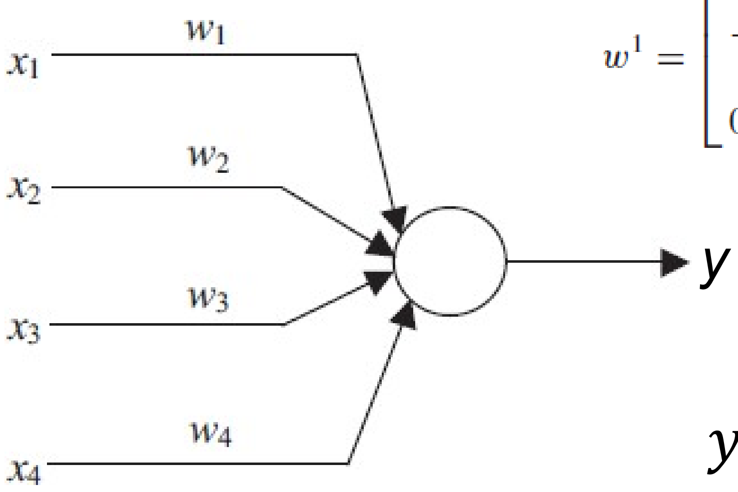
$$\frac{de}{dw} = \frac{de}{dy}\frac{dy}{dw} = -(d - y)x$$

5. Update the weighs using steepest descent:

$$w = w - \alpha \frac{\partial e}{\partial w_i}$$

Define $\Delta w = -\frac{\alpha \partial e}{\partial w_i}$ then $w = w + \Delta w$

EXAMPLE



$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix},\ x^1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix},\ x^2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} \text{ and } x^3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

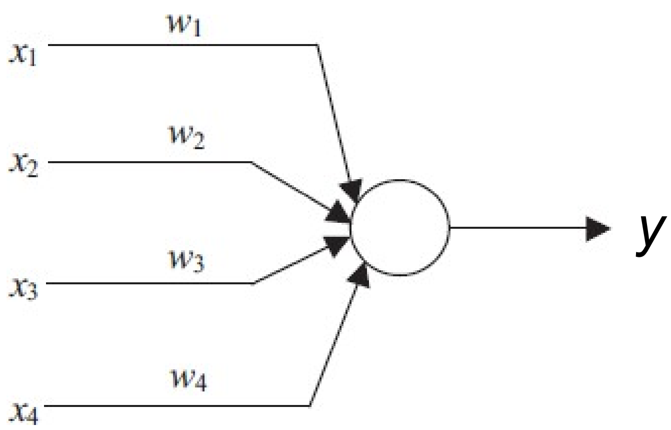
Desired output, d = [1 -1 0]

$$y = f(net) = net$$

Use ***Widrow-Hoff*** learning to update the weights

Let $\alpha = 1$

EXAMPLE



$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix},\ x^1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix},\ x^2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} \text{ and } x^3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

Desired output, d = [*1* -1 0]

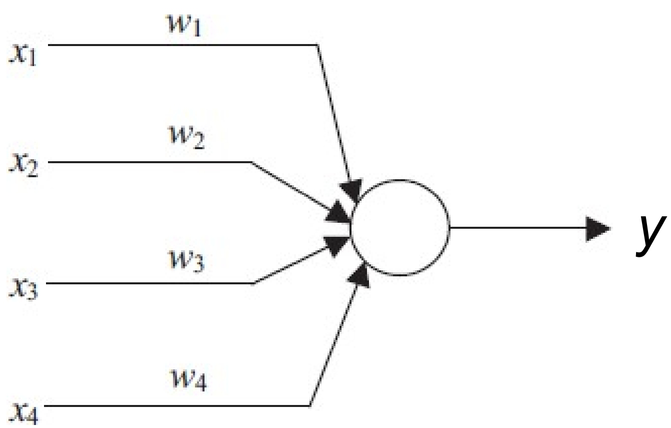
Iteration One

$$net^1 = w^1 x^1 = \begin{bmatrix} 1 & -1 & 0 & .5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = 3$$ ➔ y = 3

$$\Delta w^1 = \alpha\,(d^1 - y^1)\,x^1 = 1 * (1-3) \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} -2 \\ 4 \\ -3 \\ 0 \end{bmatrix}$$

$$w^2 = w^1 + \Delta w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + \begin{bmatrix} -2 \\ 4 \\ -3 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \\ -3 \\ 0.5 \end{bmatrix}$$

EXAMPLE



$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix},\ x^1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix},\ x^2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} \text{ and } x^3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

Desired output, d = [1 ***-1*** 0]

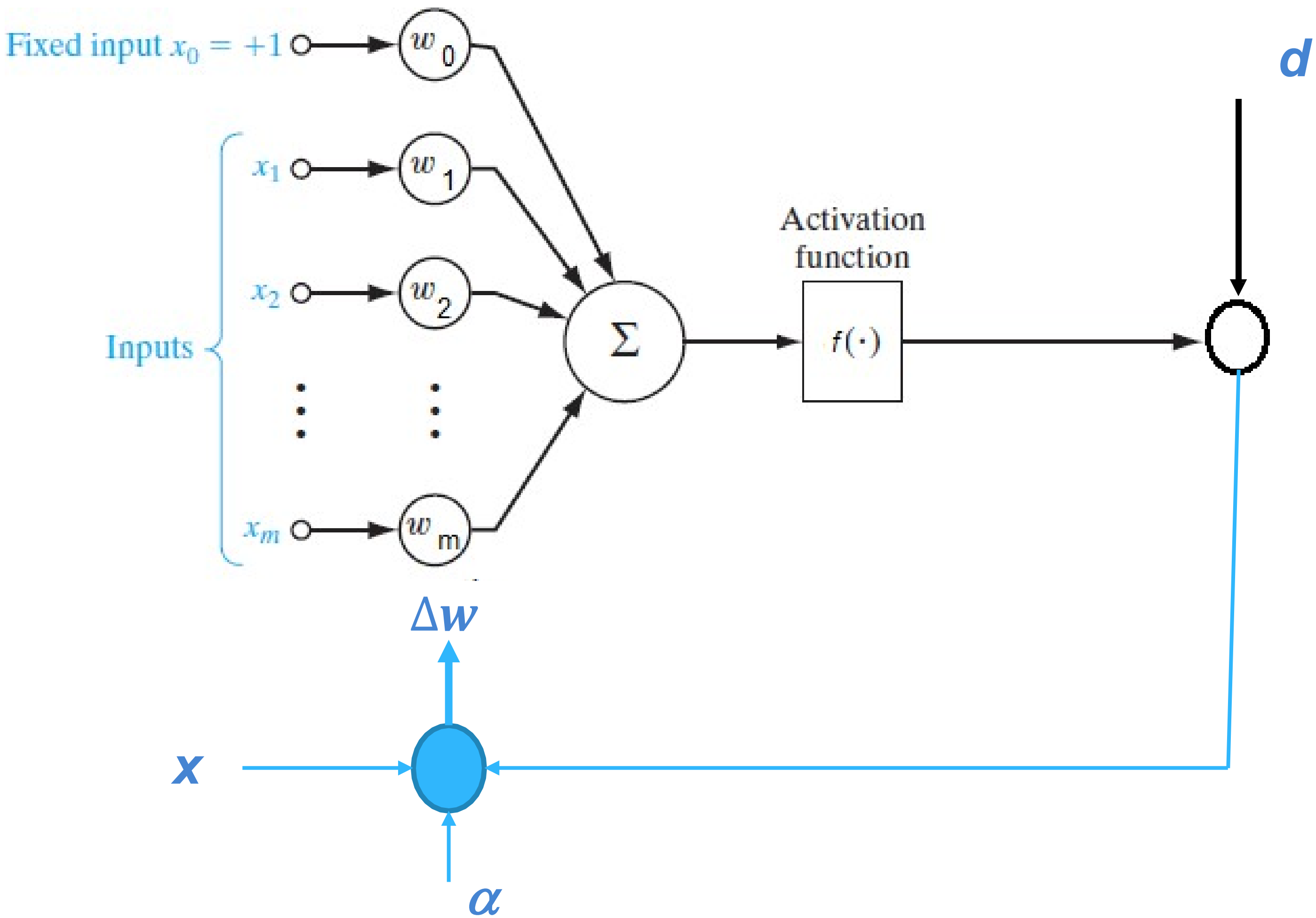
Iteration Two

$$net^2 = [-1 \quad 3 \ -3 \quad 0.5]\begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = 2.75 \Rightarrow y^2 = 2.75$$

$$\Delta w^2 = \alpha(d^2 - y^2)x^2 = 1\,(-1 - 2.75\,)\begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = \begin{bmatrix} -3.75 \\ 1.875 \\ 7.5 \\ 5.62 \end{bmatrix}$$

$$w^3 = w^2 + \Delta w^2 = \begin{bmatrix} -1 \\ 3 \\ -3 \\ 0.5 \end{bmatrix} + \begin{bmatrix} -3.75 \\ 1.875 \\ 7.5 \\ 5.62 \end{bmatrix}$$

DELTA RULE LEARNING



DELTA RULE LEARNING: SINGLE NEURON

$$net = w^{\mathrm{T}}x$$

Using a linear activation function $y = f(\text{net})$

The error between the network output and the desired output is

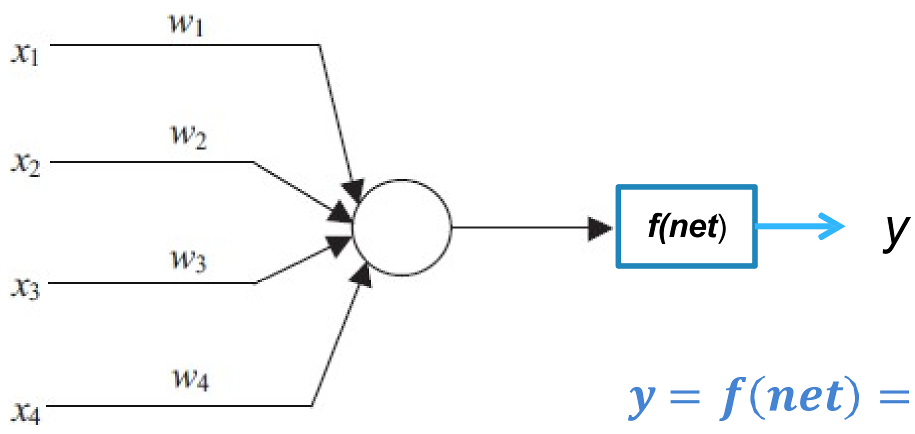
$$E = \frac{1}{2}(d - y)^2$$

The derivative w.r.t. weights is $$\frac{dE}{dw_i} = \frac{dE}{dy}\frac{dy}{dw_i} = -(d - y)f'(net)\,x_i$$

Update the weighs using delta rule $$w_i = w_i - \alpha\frac{dE}{dw_i}$$

In vector format: $W = W - \alpha\,\nabla E$

EXAMPLE



$$y = f(net) = \frac{1 - e^{-net}}{1 + e^{-net}}$$

$$Note: y' = 0.5(1 - y^2)$$

$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}, x^1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, x^2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \text{ and } x^3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

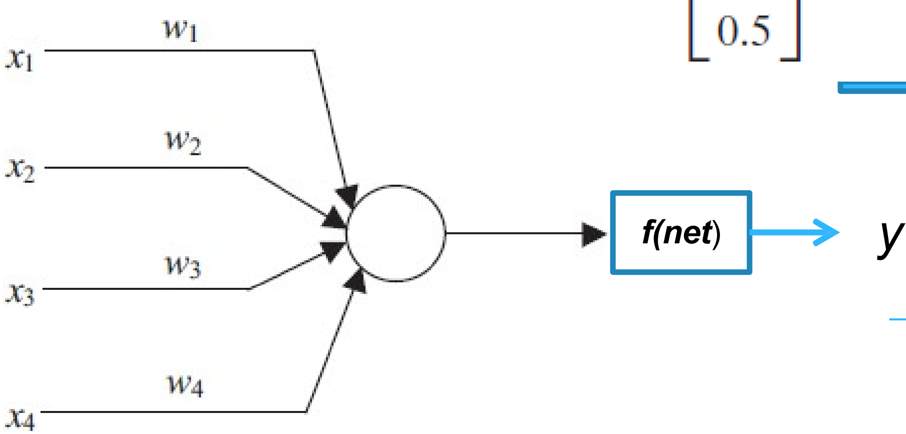
Desired output, d = [-1 -1 1]

Use **Delta Rule** learning to update the weights

with $\alpha = 0.1$

EXAMPLE

$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}, x^1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, x^2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \text{ and } x^3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$



Desired output, d = [-1 -1 1]

$$\nabla E = -\alpha\,(d - y\,)\,y'\,x$$

$$= -0.1\,(-1\; - 0.848)(0.14)\begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}$$

$$= 0.0259\begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.0259 \\ -0.0518 \\ 0 \\ -0.0259 \end{bmatrix}$$

$$w = w - \nabla E = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 0.0259 \\ -0.0518 \\ 0 \\ -0.0259 \end{bmatrix} = \begin{bmatrix} 0.9741 \\ -0.9482 \\ 0 \\ 0.5259 \end{bmatrix}$$

Iteration One
Pattern One

$$net = w^T x = [\,1 \quad -1 \quad 0 \quad 0.5]\begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2.5$$

$$y = \frac{1 - e^{-net}}{1 + e^{-net}} \Rightarrow y = 0.848$$

$$y' = 0.5\,(1 - \;y^2\,) = 0.140$$

EXAMPLE

$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}, x^1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, x^2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \text{ and } x^3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

Desired output, d = [-1 -1 1]

Pattern Two

$$net = w^T x$$

$$= [\,0.974 \quad -0.948 \quad 0 \quad 0.5259] \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}$$

$$= -1.948$$

$$y = \frac{1 - e^{-net}}{1 + e^{-net}} \Rightarrow y = -0.7505$$

$$y' = 0.5\,(1 - y^2) = 0.2184$$

$$\nabla E = -\alpha\,(d - y)\,y' x$$

$$= -0.1\,(-1 + 0.7505)(0.2184) \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0.0082 \\ -0.0027 \\ -0.0054 \end{bmatrix}$$

$$w = w - \nabla E = \begin{bmatrix} 0.974 \\ -0.948 \\ 0 \\ 0.5259 \end{bmatrix} - \begin{bmatrix} 0 \\ 0.0082 \\ -0.0027 \\ -0.0054 \end{bmatrix} = \begin{bmatrix} 0.9741 \\ -0.9563 \\ 0.0027 \\ 0.5314 \end{bmatrix}$$

EXAMP
LE

Iteration One

Pattern One

Pattern Two

Pattern Three

Iteration Two

Pattern One

Pattern Two

Pattern Three

Iteration Three

Pattern One

Pattern Two

Pattern Three

…

Two Loops

Iteration Loop

Pattern Loop

Calculate the Error

MATLAB CODE

- **% Delta Rule Example for single neuron**
- w=[1 -1 0 0.5]';
- x1=[1 -2 0 -1]'; x2=[0 1.5 -0.5 -1]'; x3=[-1 1 0.5-1]'; d1 = -1; d2=-1; d3=1;
- a=0.1;
- **for iter = 1:100**

- **% Pattern 1**
- net = w'*x1;
- y1 = (1 - exp(-net)) / (1 + exp(-net)); yp = 0.5 * (1 - y1^2);
- dE = -a * (d1 - y1)*yp*x1;
- w = w - dE;

- **% Pattern 2**
- net = w'*x2;
- y2 = (1 - exp(-net)) / (1 + exp(-net)); yp = 0.5 * (1 - y2^2);
- dE = -a * (d2 - y2)*yp*x2;
- w = w - dE;

% Pattern 3

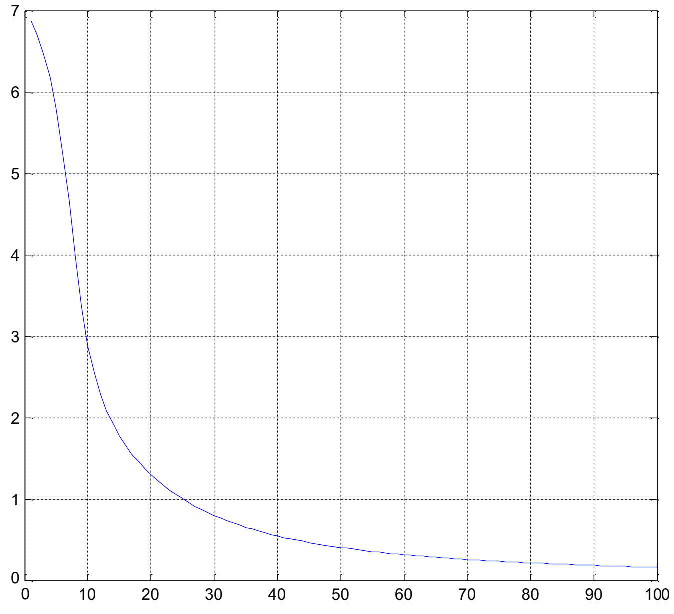
```
net = w'*x3;
y3 = ( 1 - exp(-net) ) / ( 1 + exp(-net) );
yp = 0.5 * ( 1 - y3^2);
dE = -a * (d3 - y3)*yp*x3;
w = w - dE;

Err(iter) = (d1-y1)^2 + (d2-y2)^2 + (d3-y3)^2;
```

End

```
plot(Err); grid
```

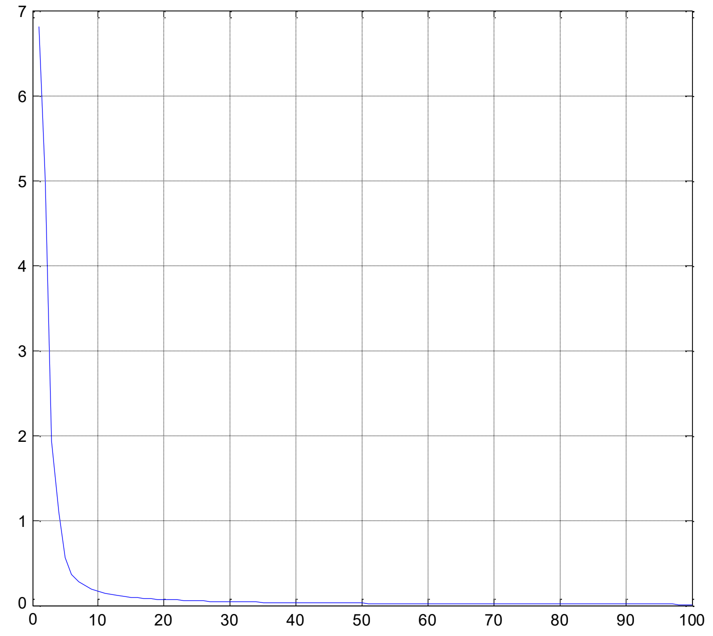
MATLAB RESULTS

$\alpha = 0.1$



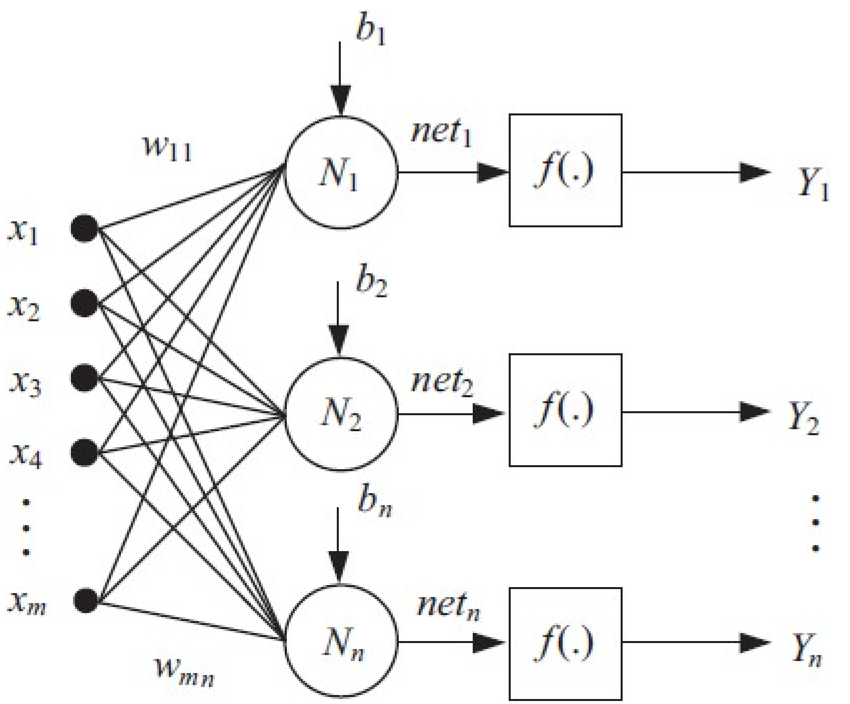
y1 = -0.8897
y2 = -0.7191
y3 = 0.7319

$\alpha = 0.9$



y1 = -0.9669
y2 = -0.9240
y3 = 0.9278

MULTI-NEURONS



$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \qquad W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & \\ \vdots & \vdots & \vdots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \qquad Y = f(W^{\top} \cdot x + b)$$

DELTA RULE LEARNING: MULTI-NEURONS

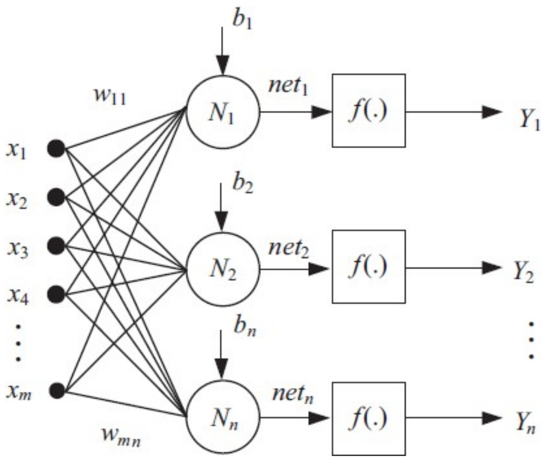
For each output neuron j = 1 : m

$$net_j = \sum_{i=1}^{m} w_{ij}\, x_i$$

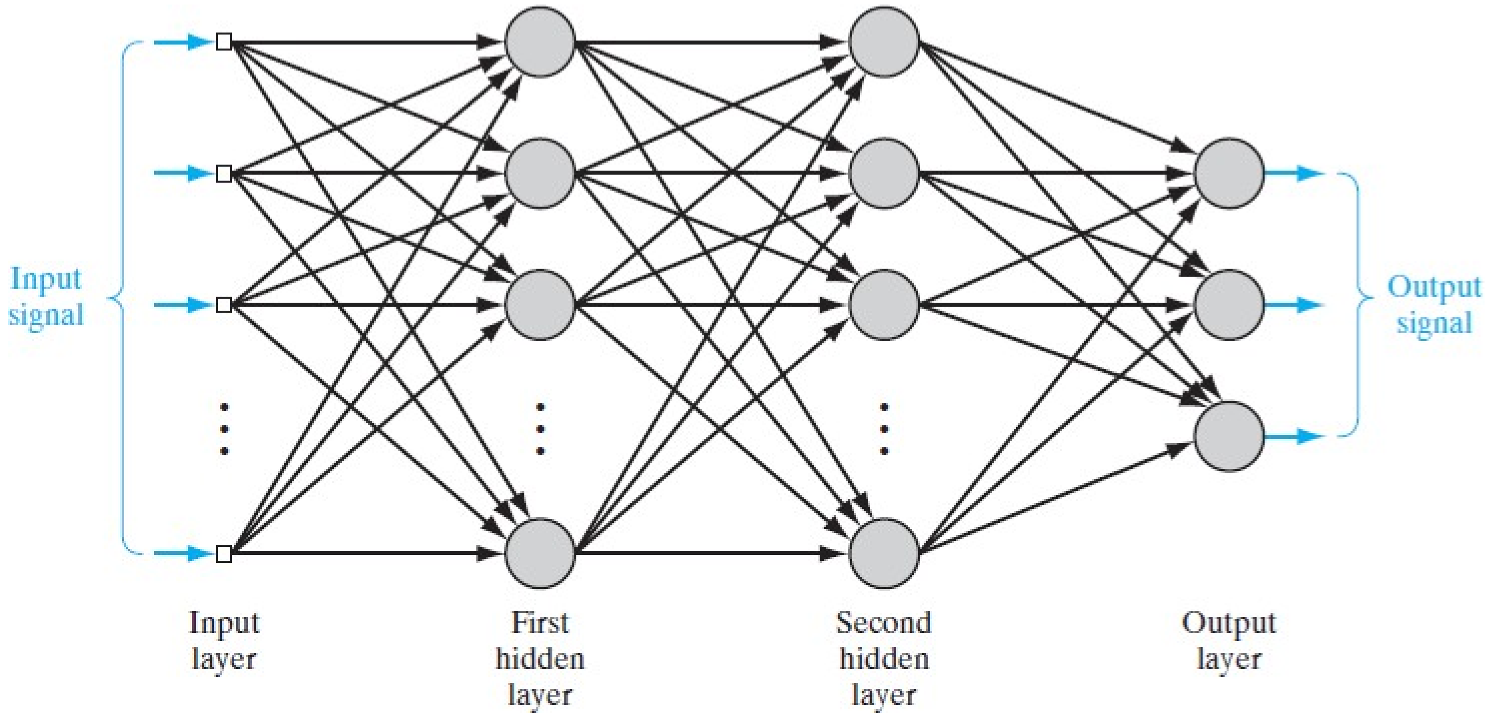
$$y_j = f\left(net_j\right)$$

$$\frac{dE}{dw_{ij}} = -\left(d_j - y_j\right) f'(net_j)\, x_i$$

$$w_{ij} = w_{ij} - \alpha \frac{dE}{dw_{ij}}$$



MULTILAYER FEEDFORWARD NETWORK



CONCLUSIONS

- Widrow-Hoff learning algorithm is a simple supervised learning algorithm used for Perceptron

- The weight change at each iteration by minimizing an error function between the perceptron output and the desired output

- Steepest descent algorithms are iterative procedures that are used to find the minimum of functions.

- These algorithms update the variables in the direction of the negative derivative (for single variable) or gradient (for multi-variables)